JOURNAL OF COMPUTER AND SYSTEM SCIENCES 26, 392–409 (1983)

Lower Bounds for the Cycle Detection Problem*

FAITH E. FICH

Computer Science Division,
University of California, Berkeley, California 94720

Received March 15, 1982; revised November 22, 1982

Given a function f over a domain and an element x in the domain, the cycle detection problem is to find a repetition in the sequence of values $x, f(x), f(f(x)), f^3(x),...$, if one exists. This paper investigates lower bounds on the number of function evaluations needed when there is a bound on the amount of memory available. For certain restricted classes of algorithms which use two memory locations optimality is achieved. A summary of the major results appears in the final section.

1. INTRODUCTION

Let f be an arbitrary function with the same domain and codomain D and let $\in D$. The cycle detection problem is to find two different nonnegative integers i and such that $f^i(x) = f^j(x)$, if they exist.

To insure maximum generality for possible choices of D and f, the manner in hich algorithms may obtain information about both is restricted. The function f is iewed as a black box which, when given an element of D as input, will produce nother as output. The only fact initially assumed about D is that it contains the lement x. Additional elements of D are made known solely as a result of function valuations. Two elements of D can be compared to determine if they are equal, but ther tests which presuppose additional structure on D are not allowed.

An algorithm for cycle detection can also be viewed as a partial decision procedure or the ultimate periodicity of any sequence

$$x, \quad f(x), \quad f(f(x)), \quad f^3(x),....$$

his is because $f^i(x) = f^j(x)$ implies $f^{i+m}(x) = f^{j+m}(x)$ for all $m \geqslant 0$. Note that if D s finite, then the pigeonhole principle implies that this sequence is always ultimately eriodic.

Now suppose this sequence is ultimately periodic. Let $t = t(f, x)$ be the index of the irst value in the sequence which is a repetition of an earlier value. Then t is the inimum number of function evaluations needed to find an i and a j given an nlimited number of storage locations. This lower bound is obtained by Sedgewick

* This work was supported by a Natural Science and Engineering Research Council of Canada ostgraduate Scholarship and by National Science Foundation Grant MCS 79–15763.

022-0000/83 $3.00

and Szymanski in [8]. The straightforward algorithm which successively computes $f^i(x)$ for $i = 0, 1, 2,...$, and compares each new value with all previous values performs exactly t function evaluations and thus achieves the lower bound.

The situation becomes more interesting when there are only M memory locations available in which to store elements of D. This bound on space is not assumed to include any temporary storage which may be required for the computation of a function value. However, it does include the memory location needed to store the starting value x, if the algorithm wants to retain it.

Focus attention on one such cycle detection algorithm. Let $t' = t'(f, x)$ denote the number of function evaluations performed by the algorithm for any particular choice of f and x. The complexity of the algorithm is measured by $\sup(t'/t)$, where the supremum is taken over all functions f and starting values x such that sequence

$$x, \quad f(x), \quad f(f(x)), \quad f^3(x),...$$

is ultimately periodic.

The cycle detection problem is mentioned by Knuth [5, pp. 4, 7, 8] in connection with the assessment of random number generators. Pollard [7] and Brent's [2] Monte Carlo algorithms for the factorization of integers involve the detection of cycles using two memory locations.

Another application is related to programs which compute the paths of orbiting bodies. Kahan [4] wanted to find examples where roundoff error caused the orbit to be nonperiodic. Cycle detection was useful for quickly eliminating from consideration those examples which resulted in periodic orbits. Algorithms which have a bound on the number of memory locations they use are particularly important for calculator implementation.

The cycle detection problem is also discussed by Sedgewick and Szymanski [8], and Sedgewick *et al.* [9]. There, attention is focused on the trade-off between the number of function evaluations and the number of comparisons performed by cycle detection algorithms.

2. An Equivalent Problem

Another characterization of the cycle detection problem can be formulated as follows. Consider placing M markers on a number line for the nonnegative integers. Initially, all markers rest on location 0. At any point in time, those locations on the number line with one or more markers on them will be referred to as *marked locations*. The markers will correspond to memory locations. A marker on location i denotes that the value $f^{(i)}(x)$ is stored in the corresponding memory location.

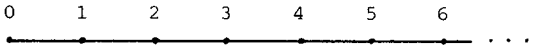

FIG. 2.1. The number line.

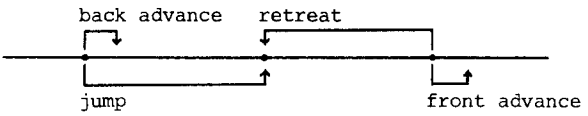

FIG. 2.2. Types of moves.

There are four allowable types of moves. If the front pile of markers is at location i.e., if n is the highest numbered marked location), then moving a marker from n to + 1 is called a *front advance.* Moving any other marker forward one position is own as a *back advance.* A *jump* consists of picking up a marker and putting it wn on top of any other marker. The special case of a jump in which there is only e marker in the front pile and this marker is moved (necessarily to a lower mbered point) is referred to as a *retreat.* A retreat is the only move that causes the lex of the location of the front pile to become smaller.

An advance corresponds to performing an instruction of the form $y \leftarrow f(y)$, which /olves one function evaluation, while a jump corresponds to a simple assignment ıtement such as $y \leftarrow z$. More complicated instructions can be broken down into ıuences of statements of these two elementary types. For example, $y \leftarrow f(z)$ can be written as $y \leftarrow z$; $y \leftarrow f(y)$. Consequently, each advance that is performed costs one it and jumps are free. Pippenger [6] observed that this model of computation is)sely related to pebbling.

The actual problem can be described now. An adversary secretly chooses two egers l and t such that $0 \leqslant l < t$. An algorithm performs moves one at a time. Its al is to reach a position such that there are two marked locations i and j such that $i < j$ and $j - i$ is an integral multiple of $t - l$. Note, in particular, that $j \geqslant t$. The :ations of the M markers at this point in time will be called an (l, t) *stopping nfiguration.* Until the algorithm reaches an (l, t) stopping configuration, it essenlly gets no information about the values of l and t. Thus the sequence of moves ıt the algorithm performs must be independent of the values of l and t chosen by : adversary. Different l's and t's merely define possibly different stopping conurations.

For any particular choices of l and t, the number of advances the algorithm :forms before stopping will be denoted by $t' = t'(t, l)$. The aim is to minimize sup $/t$), where the supremum is taken over all choices for l and t.

If t is the smallest nonnegative integer such that there exists a (necessarily unique) nnegative integer l less than t satisfying $f^l(x) = f^t(x)$, then for $j > i \geqslant 0$, $f^i(x) = x$) if and only if $i \geqslant l$ and $j - i$ is a multiple of $t - l$. Thus the quantity $\sup(t'/t)$ is :ually the complexity of the corresponding cycle detection algorithm. Throughout : rest of this paper, these two equivalent problems will be used interchangeably.

3. Lower and Upper Bounds for Any Fixed Number of Memory Locations

The lower and upper bounds obtained in this section are valid for any finite number M of memory locations or markers, provided $M \geqslant 2$. The following technical lemma is the key to the lower bound results.

LEMMA 3.1. *Let $0 \leqslant a_1 \leqslant a_2 \leqslant \cdots \leqslant a_M$ denote the locations of M markers on the number line, let $n = \max(\{a_1 + 1\} \cup \{a_k - a_{k-1} \mid 2 \leqslant k \leqslant M\})$, and let $t \geqslant a_M$. If $n > 1$, then, for any sequence of moves containing fewer than $n - 1 + t - a_M$ advances, there exists l, such that $0 \leqslant l < t$ and $l \neq a_1, \ldots, a_M$, for which no (l, t) stopping configuration is achieved during the execution of these moves.*

Proof. The value l will be chosen to be equal to 0 or to be located between the most widely spaced pair of adjacent markers. If one plus the distance from 0 to the leftmost marker is at least as large as the distances between adjacent markers, $l = 0$ is chosen. Otherwise, the particular choice for l depends on the sequence of moves performed.

More formally, if $n = a_1 + 1$, let $l = 0$. Then the largest distance between any two markers is $a_M - a_1 = t - l - (n - 1 + t - a_M)$. Since no move can cause the smallest numbered marked location to decrease, at least $n - 1 + t - a_M$ more front advances must be performed before the distance between the largest and smallest numbered marked locations is big enough for an (l, t) stopping configuration to be achieved. Note that $n > 1$ implies that $a_1 > 0 = l$.

Otherwise, $n = a_k - a_{k-1}$ for some k such that $2 \leqslant k \leqslant M$. Let α be the largest numbered location among all those which contained a marker at any time during the execution of the sequence of moves.

If $\alpha < t$, then no location t or larger contained a marker. Thus, for all l such that $0 \leqslant t < t$, no (l, t) stopping configuration could have occurred. In this case, let $l = a_{k-1} + 1$. Note that $l < a_k$, since $n > 1$.

Therefore suppose $\alpha \geqslant t$. In this case, let $l = a_{k-1} + n - 1 + t - \alpha = a_k - 1 + t - \alpha$. Since $\alpha \leqslant a_M + (n - 2 + t - a_M) = n - 2 + t$, it follows that $a_{k-1} + 1 \leqslant l \leqslant a_k - 1$. Consider any location j which contained a marker at some point during the execution of the sequence of moves. Note that $j \leqslant \alpha$. At least $\alpha - a_M$ of the at most $n - 2 + t - a_M$ advances performed were involved in moving the location of the front marker from a_M to α. Therefore, during this time, all marked locations less than a_k must have been less than or equal to $a_{k-1} + n - 2 + t - \alpha < l$. In particular, if i is an integer such that $j > i \geqslant l$ and $j - i = d(t - l)$ for some integer $d \geqslant 1$, then $l \leqslant i \leqslant j - (t - l) = (j - \alpha) + a_k - 1 \leqslant a_k - 1$, so that location i could not have contained a marker throughout this entire period of time. Hence, no (l, t) stopping configuration is achieved. ∎

It is easy to see that if $n = \max(\{a_1 + 1\} \cup \{a_k - a_{k-1} \mid 2 \leqslant k \leqslant M\}) = 1$ and $t > a_m$, then, starting at the configuration $a_1 \leqslant \cdots \leqslant a_M$, at least $n - 1 + t - a_M$ more

front advances must be performed before an (l, t) stopping configuration is achieved for any l such that $0 \leqslant l < t$.

THEOREM 3.2. *For any cycle detection algorithm which uses at most M locations and which performs at least one retreat,* $\sup(t'/t) \geqslant 1 + (1/(M-1))$.

Proof. Let $0 \leqslant a_1 \leqslant a_2 \leqslant \cdots \leqslant a_M$ denote the locations of the M markers just after the first retreat is performed and let $n = \max(\{a_1 + 1\} \cup \{a_k - a_{k-1} | 2 \leqslant k \leqslant M\})$. Let a denote the location of the front pile before the retreat took place. To achieve this state of affairs, a front advances must have been performed.

First, suppose that at least one back advance was also performed. Let $t = a + 1$. Then the algorithm performed at least t advances to achieve the configuration $a_1 \leqslant \cdots \leqslant a_M$. Prior to the retreat, all markers occupied locations with index less than t. It follows that, for any l such that $0 \leqslant l < t$, no (l, t) stopping configuration could have occurred. Lemma 3.1 and the remark which follows it imply that there exists l, such that $0 \leqslant l < t$, for which the algorithm must perform at least $n - 1 + t - a_M$ more advances before stopping. For this choice of l, $t' \geqslant t + n - 1 + t - a_M$.

Now suppose that no back advances were performed to achieve the configuration $a_1 \leqslant \cdots \leqslant a_M$. Further suppose that the move prior to the retreat was a front advance.

If $n > 1$, let $t = a$. Note that (l, t) stopping configurations have previously occurred only for $l = a_1, \ldots, a_M$. Since Lemma 3.1 guarantees the existence of l such that $0 \leqslant l < t$, $l \neq a_1, \ldots, a_M$, and for which the algorithm must perform at least $n - 1 + t - a_M$ more advances before achieving an (l, t) stopping configuration, $t' \geqslant t + n - 1 + t - a_M$.

If $n = 1$ and $a > a_M + 1$, let $t = a$ and $l = a_M + 1$. In this case, no (l, t) stopping configuration has already been achieved and at least $n - 1 + t - a_M$ more front advances must be performed to achieve one. Hence $t' \geqslant t + n - 1 + t - a_M$.

Since the last move performed to achieve the configuration $a_1 \leqslant \cdots \leqslant a_M$ was a retreat, $a_r = a_{r-1}$ for some r, $2 \leqslant r \leqslant M$. Hence $n(M-1) \geqslant a_1 + 1 + \sum_{k=2}^{r-1} (a_k - a_{k-1}) + \sum_{k=r+1}^{M} (a_k - a_{k-1}) = a_1 + 1 + \sum_{k=2}^{M} (a_k - a_{k-1}) = a_M + 1$. Also note that $M \geqslant 2$ and $t \geqslant a \geqslant a_M + 1$. Therefore, for the three cases discussed above,

$$\frac{t'}{t} \geqslant 1 + \frac{n - 1 + t - a_M}{t} \geqslant 1 + \frac{\dfrac{a_M + 1}{M-1} - 1 + t - a_M}{t}$$

$$= 1 + \frac{t + (M-2)(t - a_M - 1)}{(M-1)\,t} \geqslant 1 + \frac{1}{M-1}.$$

In all other cases, let $t = a + 1$. Then, for any l such that $0 \leqslant l < t$, no (l, t) stopping configuration could have occurred on or before the first retreat. By Lemma 3.1 and the remark which follows it, there exists l such that $0 \leqslant l < t$ and $t' \geqslant a + n - 1 + t - a_M = 2t + n - 2 - a_M$.

Let m be the number of marked locations in the configuration $0 \leqslant a_1 \leqslant \cdots \leqslant a_M$. Then $nm \geqslant a_1 + 1 + \sum_{k=2}^{M} (a_k - a_{k-1}) = a_M + 1$. Note that all marked locations are between 0 and a_M; therefore $m \leqslant a_M + 1$. Hence $a_M \leqslant M - 3$ implies $m \leqslant M - 2$. If the move performed by the algorithm prior to the first retreat was a jump, then $m \leqslant M - 2$. In these two cases,

$$\frac{t'}{t} \geqslant 2 + \frac{\dfrac{a_M + 1}{m} - 2 - a_M}{t}$$

$$= 2 - \frac{(M-2)(a_M+2)}{(M-1)\,t} + \frac{(M-1-m)\left(\dfrac{a_M+1}{m}\right) - 1}{(M-1)\,t} \geqslant 2 - \frac{M-2}{M-1}$$

$$= 1 + \frac{1}{M-1}$$

since $a_M + 2 \leqslant a + 1 = t$ and $(M - 1 - m)((a_M + 1)/m) \geqslant 1$.

The only remaining case to consider is when $n = 1$, $a = a_M + 1$, and $a_M = M - 2$. (Note that $a_M > M - 2$ implies $n > 1$.) Consider the locations of the markers the first time the location t is marked. Since $t = M$, there exists a location $l < t$ which does not contain a marker. For this choice of l, at least one more advance is required to achieve an (l, t) stopping configuration. Hence

$$\sup \frac{t'}{t} \geqslant \frac{a + (n - 1 + t - a_M) + 1}{t} = 1 + \frac{2}{M} \geqslant 1 + \frac{1}{M-1}. \qquad \blacksquare$$

THEOREM 3.3. *For all cycle detection algorithms which use at most M locations,* $\sup(t'/t) \geqslant 1 + (1/(M-1))$. *Furthermore, if an algorithm has a preset bound on the number of jumps it performs, then* $\sup(t'/t) \geqslant 1 + \sqrt{2}$.

Proof. Suppose the algorithm uses arbitrarily many jumps. Let $0 \leqslant a_1 \leqslant a_2 \leqslant \cdots \leqslant a_M$ denote the locations of the M markers immediately after a jump has occurred. Clearly, at least a_M front advances are required to achieve this configuration.

Let $t = a_M + 1$. Because of Theorem 3.2, it may be assumed that the algorithm never performs retreats. Hence no location greater than or equal to t has ever been marked. In particular, this implies that for any l, $0 \leqslant l < t$, the algorithm has not already achieved an (l, t) stopping configuration. By Lemma 3.1 and the remark which follows it, there exists an l, $0 \leqslant l < t$, for which the algorithm must perform at least $n - 1 + t - a_M$ more advances before halting, where $n = \max(\{a_1 + 1\} \cup \{a_k - a_{k-1} \mid 2 \leqslant k \leqslant M\})$.

Since $a_r = a_{r-1}$ for some r, $2 \leqslant r \leqslant M$, it follows that $n(M-1) \geqslant a_1 + 1 + \sum_{k=2}^{M} (a_k - a_{k-1}) = a_M + 1 = t$. Thus

$$\frac{t'}{t} \geqslant \frac{a_M + (n-1+t-a_M)}{t} = 1 + \frac{n}{t} - \frac{1}{t} \geqslant 1 + \frac{n}{n(M-1)} - \frac{1}{t}$$
$$= 1 + \frac{1}{M-1} - \frac{1}{a_M + 1}.$$

For the algorithm to work correctly, the front marker must move arbitrarily far along the number line. Since no retreats are performed, the location of this marker can never decrease. Also, the algorithm uses arbitrarily many jumps. Thus a_M, the location of the front marker immediately after a jump has occurred, must become arbitrarily large. Hence $\sup(t'/t) \geqslant 1 + (1/(M-1))$.

On the other hand, suppose the algorithm has a preset bound on the number of jumps it performs. Let $0 \leqslant b_1 \leqslant b_2 \leqslant \cdots \leqslant b_M$ denote the locations of the M markers after the last jump is performed. (If the algorithm uses no jumps, let $0 = b_1 = b_2 = \cdots = b_M$.) At least b_M front advances must be performed to reach this state. From this point on, each marker can only be moved via advances and hence to higher numbered locations.

Let $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_M$ be the locations of the markers at some later point in the algorithm. Note that $a_k \geqslant b_k$ for $1 \leqslant k \leqslant M$. To get to this configuration from the other one requires a total of $\sum_{k=1}^{M} (a_k - b_k)$ advances.

Since l can be arbitrarily large, the front two markers must both travel arbitrarily far along the number line. Therefore we may assume, without loss of generality, that a_{M-1} is larger than any location marked by the algorithm prior to its last jump.

The idea of the remainder of the proof is to have the adversary's choice for l and t depend on the relative positions of a_M and a_{M-1}. If the distance between a_M and a_{M-1} is large and t is chosen to be small, then many of the front moves performed by the algorithm will have been "wasted." On the other hand, if a_M is not very far ahead of the other markers, then a strategy suggested by Lemma 3.1 yields good results.

If $a_M \geqslant \sqrt{2}a_{M-1} + 1 + 2\sqrt{2} + b_{M-1}$, let $l = a_{M-1} + 1$ and $t = a_{M-1} + 2$. Up to and including the point in the algorithm when the configuration $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_M$ is obtained, all markers except the front one have been at locations less than or equal to $a_{M-1} < l$. Hence the algorithm could not have achieved an (l, t) stopping configuration during this period of time. Thus $t' \geqslant 1 + b_M + \sum_{k=1}^{M} (a_k - b_k) \geqslant 1 + a_M + a_{M-1} - b_{M-1} \geqslant (1 + \sqrt{2})(a_{M-1} + 2)$ and $(t'/t) \geqslant 1 + \sqrt{2}$. Therefore suppose $a_M < \sqrt{2}a_{M-1} + 1 + 2\sqrt{2} + b_{M-1}$ for all configurations $0 \leqslant a_1 \leqslant a_2 \leqslant \cdots \leqslant a_M$ under consideration.

Let $t = a_M + 1$, a location which has never contained a marker. If $M > 2$, then, by Lemma 3.1 and the remark which follows it, at least $a_{M-1} - a_{M-2}$ more advances must be performed by the alogrithm before an (l, t) stopping configuration is achieved. Thus $t' \geqslant b_M + \sum_{k=1}^{M} (a_k - b_k) + a_{M-1} - a_{M-2} \geqslant (a_{M-2} - b_{M-2}) + (a_{M-1} - b_{M-1}) + a_M + (a_{M-1} - a_{M-2}) = a_M + 2a_{M-1} - b_{M-2} - b_{M-1} > (1 + \sqrt{2})(a_M + 1) - C$, where $C = 5 + 2\sqrt{2} + (1 + \sqrt{2})\, b_{M-1} + b_{M-2}$. Similarly, if $M = 2$, then at least $a_{M-1} + 1$ more advances are required and $t' \geqslant b_M + (a_{M-1} - b_{M-1}) + (a_M - b_M) + a_{M-1} + 1 > (1 + \sqrt{2})(a_M + 1) - C$, where $C = 4 + 2\sqrt{2} + (1 + \sqrt{2})\, b_{M-1}$. Hence $(t'/t) > 1 + \sqrt{2} - (C/(a_M + 1))$ for some constant C.

If the algorithm is to work correctly, then for any choice of t there must be a point in the algorithm where $a_M \geqslant t$. Since t can be arbitrarily large, $\sup\{1+\sqrt{2} - (C/(a_M+1))\} = 1+\sqrt{2}$. Thus $\sup(t'/t) \geqslant 1+\sqrt{2}$. ∎

Suppose $0 \leqslant l < t$ and consider the point at which a given algorithm stops. The last move cannot have been a jump because, otherwise, the set of distances between the marked locations would just be a subset of the set of distances for the configuration occurring immediately before the jump. This fact is used in the proof of the following theorem.

THEOREM 3.4. *For any cycle detection algorithm that performs no back advances, there is a cycle detection algorithm that performs no back advances or retreats and has complexity at least as small.*

Proof. To make the exposition clearer, the terminology *recovery point* of a retreat will be used to denote the highest numbered marked location immediately after the retreat has been performed. Just before the retreat is performed, the recovery point is the second highest numbered marked location.

Suppose we are given an algorithm which uses no back advances, but does use retreats. Without loss of generality, we may assume that every retreat in the algorithm is a jump taking the front marker to its recovery point. This is because a retreat to any other marked location can be achieved by first retreating to the recovery point and then jumping from the recovery point to the desired location. Note that this second jump is not classified as a retreat because, after it is performed, at least one marker still remains on the recovery point.

A new algorithm is constructed from the original algorithm as follows: For every front advance in the original algorithm there is a corresponding front advance in the new algorithm. If, in the original algorithm, there is a jump moving the rth marker on top of the sth marker and it is not a retreat, then, in the new algorithm, there is a jump moving the rth marker on top of the sth marker. In the case of a retreat, instead of moving the one marker in the front pile to the recovery point, all the markers on the second highest numbered location (i.e., the location that would have been the recovery point had a retreat been performed) are moved (via jumps) to the front pile.

Let $a_1 \leqslant \cdots \leqslant a_M$ denote the locations of the M markers at some point during the execution of the original algorithm and let $a'_1 \leqslant \cdots \leqslant a'_M$ denote the corresponding configuration in the new algorithm. Then it is easy to show by induction that $a'_i \geqslant a_i$ and $a'_M - a'_i \geqslant a_M - a_i$ for $1 \leqslant i \leqslant M$.

Now let $a_1 \leqslant \cdots \leqslant a_M$ denote the locations of the M markers when the original algorithm stops, for some choice of l and t. Since this algorithm uses no back advances, it follows from the discussion preceding this theorem that the last move performed must have been a front advance. Then $a_M - a_i$ is a multiple of $t-l$ for some i, $1 \leqslant i \leqslant M-1$.

Consider the corresponding configuration $a'_1 \leqslant \cdots \leqslant a'_M$ of the new algorithm. Because no back advances are performed, location a'_i must have remained marked continuously since the last time it was the position of the front marker.

Now $a'_M - a'_i \geqslant a_M - a_i > 0$; therefore the front marker must reach location $a'_i + a_M - a_i$ at some time between the points when the front marker was last on a'_i and when the current configuration $a'_1 \leqslant \cdots \leqslant a'_M$ was reached. This is because the front marker advances only one square at a time. Since $a'_i \geqslant a_i \geqslant l$ and $(a'_i + a_M - a_i) - a'_i = a_M - a_i$ is a multiple of $t - l$, the new algorithm achieves an (l, t) stopping configuration at this point in time. Thus the new algorithm has complexity at least as small as the original algorithm. ∎

Therefore, in the absence of back advances, retreats are never necessary. In fact, it can be assumed that all jumps are to the front pile.

THEOREM 3.5. *For any cycle detection algorithm that performs no back advances, there is a cycle detection algorithm that performs no back advances, has the front pile as the destination of all its jumps, and has complexity at least as small.*

Proof. By Theorem 3.4, we may assume that the original algorithm performs no retreats. Hence the location of the front pile is a nondecreasing function of time.

A new algorithm is constructed as follows: Associate with each marker a virtual location, which may or may not be the same as its physical location. The idea of these virtual locations is that, at any point in time, they correspond to the locations of the markers in the original algorithm. The virtual locations of the markers will always form a subset of their physical locations (i.e., the marked locations). Thus if an (l, t) stopping configuration is achieved in the original algorithm, then one will also be achieved at the corresponding point in the new algorithm. Initially, the virtual and physical locations of all markers are identical. It will also remain true that all markers virtually located at the front pile are also physically located there.

When the original algorithm performs a front advance, the new algorithm also performs a front advance, using a marker whose virtual location is also the front pile. The new virtual location of this marker is set equal to its new physical location.

When the original algorithm performs a jump to the front pile, the new algorithm will also do so. If there is a marker whose virtual location differs from its physical location and is the source of the jump in the original algorithm, then the jump is performed using that marker. Otherwise, a marker whose virtual and physical locations are both equal to the source location of the original jump is used to perform the jump in the new algorithm. In either case, the virtual location of the chosen marker is updated to agree with its new physical location.

Finally, if the original algorithm performs a jump to some location other than the front pile, then the new algorithm will perform no corresponding move. All that happens is that one marker, virtually located at the source of the jump, will have its virtual location set to the destination of the jump. A marker whose virtual location differs from its physical location is chosen, if possible.

It is clear that the new algorithm performs no back advances, has the front pile as the destination of all its jumps, and has complexity at least as small as the original algorithm. ∎

```
b ← 1
m ← 1
for k ← 1 to M do
        begin
        location (k) ← 0
        value (k) ← x
        end
repeat
        begin
        if location (m) ≡ −1 mod b then
                if m = M then
                        begin
                        b ← 2b
                        m ← 1 + ⌊M/2⌋
                        for k ← 1 to m − 1 do
                                begin
                                location (k) ← location (2k)
                                value (k) ← value (2k)
                                end
                        location (m) ← location (M)
                        value (m) ← value (M)
                        end
                else    begin
                        location (m + 1) ← location (m)
                        value (m + 1) ← value (m)
                        m ← m + 1
                        end
        value (m) ← f (value (m))
        location (m) ← location (m) + 1
        end
until   value (i) = value (j) for some 1 ⩽ i < j ⩽ m
```

FIG. 3.1. Sedgewick and Szymanski's algorithm.

Now turn attention to the upper bound. Consider the cycle detection algorithm given in Fig. 3.1. It is a slight modification of Sedgewick and Szymanski's algorithm appearing in [8, 9].

Associated with each marker k, $1 \leqslant k \leqslant M$, are the two quantities *location* (k) and *value* (k). The former is the current position of the marker on the number line and the latter is the corresponding element in the sequence, namely, $f^{location(k)}(x)$.

This algorithm performs no back advances. The idea is to achieve configurations where the markers are evenly spaced. Each time such a configuration is attained, the distance between the markers will be a factor of two larger. Every other marked

location of an evenly spaced configuration still will be marked when the next such configuration is attained.

On each iteration the front marker (marker m) moves forward, leaving behind markers as it passes those locations which are one less than a multiple of b. When all the M markers are located before different multiples of b, there is no marker available to be left behind when the front marker next advances. In this case b is doubled in value and all markers (except for the front marker) with positions no longer congruent to $-1 \bmod b$ are treated as if they had been removed. These markers are now available to be left behind.

THEOREM 3.6. *For Sedgewick and Szymanski's algorithm,* $\sup(t'/t) = 1 + (2/(M-1))$.

Proof. Suppose $0 \leqslant l < t \leqslant (M-1)b$, where b is a power of 2. Let $k = \lceil (l+1)/b \rceil$. Since $Mb - 1 \geqslant bk - 1 + t - l > bk - 1 \geqslant l$, an (l, t) stopping configuration occurs when locations $bk - 1 + t - l$ and $bk - 1$ are both marked. This condition is achieved when the front marker reaches location $bk - 1 + t - l$. Hence $t' \leqslant bk - 1 + t - l \leqslant t + b - 1$. (Note that t' is not necessarily equal to $bk - 1 + t - l$ since the algorithm might have, in fact, stopped earlier.)

For t in the range $(M-1)2^r + 1 \leqslant t \leqslant (M-1)2^{r+1}$, it follows that $t' \leqslant t + 2^{r+1} - 1$. Thus

$$\frac{t'}{t} \leqslant \frac{t + 2^{r+1} - 1}{t} = 1 + \frac{2^{r+1} - 1}{t} \leqslant 1 + \frac{2^{r+1} - 1}{(M-1)2^r + 1}.$$

Note that $(t'/t) = 1 + ((2^{r+1} - 1)/((M-1)2^r + 1))$ when $t = (M-1)2^r + 1$ and $l = 0$. Hence

$$\sup \frac{t'}{t} = \sup_r \left\{ 1 + \frac{2^{r+1} - 1}{(M-1)2^r + 1} \right\} = 1 + \frac{2}{M-1}. \quad \blacksquare$$

From Theorem 3.3 it follows that Sedgewick and Szymanski's algorithm is very close to being optimal.

4. LOWER AND UPPER BOUNDS FOR TWO MEMORY LOCATIONS

Throughout this section attention is restricted to algorithms having only two memory locations available in which to store elements of the domain D. As a result, algorithms can be denoted by a sequence of pairs

$$(b_0, a_0), (b_1, a_1), (b_2, a_2), \ldots,$$

where b_i and a_i, $b_i \leqslant a_i$, represent the locations of the two markers after the ith move has been performed. Using this notation, move i is a front advance if $a_i = a_{i-1} + 1$

and $b_i = b_{i-1}$, a back advance if $a_i = a_{i-1}$ and $b_i = b_{i-1} + 1$, a jump in the forward direction if $a_i = b_i = a_{i-1}$, and a retreat if $a_i = b_i = b_{i-1}$.

The first pair (b_0, a_0) always has the value $(0, 0)$. If $b_i = a_i$, the next move must be a front advance. This would be the situation initially as well as just after a jump has been performed.

THEOREM 4.1. *For any cycle detection algorithm that uses two memory locations and performs retreats there is a cycle detection algorithm that uses two memory locations, performs no retreats, and has complexity at least as small.*

Proof. Consider any algorithm (b_0, a_0), (b_1, a_1), (b_2, a_2),... for the cycle detection problem. We can construct a new algorithm (b_0', a_0'), (b_1', a_1'), (b_2', a_2'),... by replacing every retreat (i.e., jump in the backward direction) by a jump in the forward direction.

It can be shown by induction that the distance between the two markers a_i' and b_i' is the same as the distance between the markers a_i and b_i. However, after the original algorithm performs its first retreat, the markers a_i' and b_i' are located farther along the number line than a_i and b_i, respectively. From these facts, it is easy to see that the complexity of the new algorithm is no larger than the complexity of the original algorithm. ∎

Because of Theorem 4.1, it can be assumed that the algorithms under consideration use no retreats. This assumption will apply for the rest of Section 4. Attention first will be focused on algorithms which use no jumps. In this case, each move represents one function evaluation. Thus if an algorithm stops after exactly i moves, then $t' = i$.

Suppose the algorithm performs $a(1)$ front advances, then $b(1)$ back advances, then $a(2)$ more front advances, followed by $b(2)$ back advances, etc. Let $A(k) = \sum_{i=1}^{k} a(i)$ and $B(k) = \sum_{i=1}^{k} b(i)$. Note that $a(i), b(i) \geqslant 1$ for all $i \geqslant 1$, so $A(k+1) > A(k)$ and $B(k+1) > B(k)$ for all $k \geqslant 1$. Furthermore, let $A(0) = 0 = B(0)$. If $B(k) + A(k) \leqslant i \leqslant B(k) + A(k+1)$, then, after i moves, exactly $B(k)$ back advances and $i - B(k)$ front advances have been performed. Thus $a_i = i - B(k)$ and $b_i = B(k)$. Similarly, if $B(k) + A(k+1) \leqslant i \leqslant B(k+1) + A(k+1)$, then $a_i = A(k+1)$ and $b_i = i - A(k+1)$.

The following technical Lemma 4.2 determines, for the situation illustrated in Fig. 4.1, when the algorithm will first achieve an (l, t) stopping configuration.

LEMMA 4.2. *If* $1 + A(k) \leqslant t \leqslant A(k+1)$ *and* $0 \leqslant l \leqslant B(k)$, *then* $t' = 2B(j) + t - l$, *where* $j = \min\{h \mid h \geqslant k \text{ and } A(h+1) - B(h) \geqslant t - l\}$.

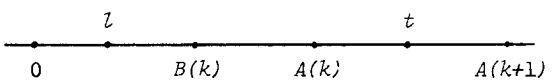

FIG. 4.1. A typical situation.

Proof. An (l, t) stopping configuration cannot be achieved before the front marker reaches t. It follows that $a_{t'} \geqslant t > A(k)$ and thus $t' > A(k) + B(k)$. Since $A(k) - B(k) < t - l$ and the back marker is already beyond l by the time $A(k) + B(k)$ moves have been performed, the algorithm will stop when the distance between the two markers reaches $t - l$. Also notice that the first (l, t) stopping configuration occurs as the result of a front advance. From these facts, it is easy to see that $b_{t'} = B(j)$, $a_{t'} = t - l - B(j)$, and, hence, $t' = 2B(j) + t - l$. ∎

THEOREM 4.3. *For any cycle detection algorithm which uses only two memory locations and performs no jumps,* $\sup(t'/t) \geqslant 3$.

Proof. Attention is focused on those points when the algorithm temporarily stops doing front advances and starts doing back advances. The relative size of the distance from 0 to the leftmost marker and the distance between the two markers determines the action of the adversary. If the first of these two quantities is small, then t is chosen to be as small as possible so that, in effect, a large number of the front advances were "wasted." Otherwise, l is chosen to be 0. The adversary must choose t large enough so that none of the previous configurations of the algorithm were (l, t) stopping configurations. Then the back advances the algorithm next performs will be "wasted." Also, the number of advances the algorithm must perform before stopping is still significant.

Suppose $2B(i) \leqslant A(i+1)$. Let $l = B(i) + 1$ and $t = B(i) + 2$.

If $j \leqslant A(i+1) + B(i)$, then $b_j \leqslant B(i) < l$. However, if $j = A(i+1) + B(i) + 1$, then $a_j = A(i+1)$, $b_j = B(i) + 1 = l$, and $a_j - b_j = A(i+1) - (B(i) + 1) = (A(i+1) - B(i) - 1)(t - l)$. Hence $t' = A(i+1) + B(i) + 1$ and $(t'/t) = (A(i+1) + B(i) + 1)/(B(i) + 2) \geqslant (2B(i) + B(i) + 1)/(B(i) + 2) = 3 - (5/(B(i) + 2))$.

Note that $\max_i B(i) = \infty$. Otherwise, the algorithm will not work for $l \geqslant 1 + \max_i B(i)$. Therefore, if $2B(i) \leqslant A(i+1)$ infinitely often, then $\sup(t'/t) \geqslant \sup_i\{3 - (5/(B(i) + 2))\} = 3$.

If $2B(i) \leqslant A(i+1)$ is not true infinitely often, then there exists $k \geqslant 0$ such that $2B(i) > A(i+1)$ for all $i \geqslant k$. In this case, let $t = 1 + A(k)$ and let $l = 0$. Then, from Lemma 4.2, $t' = 2B(j) + t - l$, where $j = \min\{h \mid h \geqslant k \text{ and } A(h+1) - B(h) \geqslant t - l\}$. In addition, $2B(j) > A(j+1)$ so that $B(j) > A(j+1) - B(j) \geqslant t - l = t$. Thus $t' = 2B(j) + t - l > 2t + t = 3t$ and hence $\sup(t'/t) \geqslant (t'/t) > 3$. ∎

Theorem 4.3 remains true even if the the algorithm is allowed to perform a fixed finite number of jumps. The ideas of the proof are essentially the same. Details can be found in [3]. Allender [1] further generalized this result. Specifically, he showed that, for any cycle detection algorithm which uses at most M locations and which performs at most a fixed finite number of jumps, $\sup(t'/t) \geqslant 3$.

Floyd's algorithm, depicted in Fig. 4.2, achieves this bound using only two locations. This well-known algorithm ([5, pp. 7, 453; 7–9]) performs only advances, with the front marker moving twice as fast as the back marker.

```
y ← x
z ← x
i ← 0
repeat
        y ← f(y)
        z ← f(f(z))
        i ← i + 1
until   y = z
j ← 2i
```

FIG. 4.2. Floyd's algorithm.

Now consider the class of algorithms which perform no back advances. Since the back marker must move arbitrarily far along the number line, such an algorithm must perform an infinite number of jumps.

For any algorithm, let c_q denote the location of the two markers after the qth jump has been performed. Let $c_0 = 0$. For $c_q + q \leqslant i \leqslant c_{q+1} + q$, it follows that $b_i = c_q$ and $a_i = i - q$. Also, if such an algorithm stops after exactly i moves, then $t' = a_i$.

The following technical lemma is similar in flavor to Lemma 4.2.

LEMMA 4.4. *If* $0 \leqslant l < t$ *and* $t > c_s$, *then* $t' = c_q + t - l$, *where* $q = \min\{j \mid j \geqslant s, l \leqslant c_j, \text{ and } t - l \leqslant c_{j+1} - c_j\}$.

Proof. An (l, t) stopping configuration cannot be achieved until after the sth jump has occurred. However, after that jump, the algorithm will stop as soon as the difference between the two markers reaches $t - l$. ∎

THEOREM 4.5. *For any algorithm which uses only two memory locations and performs no back advances,* $\sup(t'/t) \geqslant (3 + \sqrt{5})/2$.

Proof. This proof focuses on the points in the algorithm immediately preceding jumps. An adversary argument similar to the one given in Theorem 4.3 is used. Suppose $c_{p+1} \geqslant (c-1)(3+\sqrt{5})/2$. Let $l = c_p + 1$, let $t = c_p + 2$, and let k be the smallest integer such that (b_k, a_k) is an (l, t) stopping configuration. If $k \leqslant c_{p+1} + p$, then $b_k \leqslant c_p < l$, a contradiction. If $k = c_{p+1} + p + 1$, then the kth move is a jump and $a_k - b_k = 0 < t - l$, another contradiction. Hence $k \geqslant c_{p+1} + p + 2$. For $i = c_{p+1} + p + 2$, $b_i = c_{p+1} \geqslant l$, $a_i = c_{p+1} + 1$, and $a_i - b_i = 1 = t - l$. Therefore $k = c_{p+1} + p + 2$ and $t' = c_{p+1} + 1 \geqslant (c-1)(3+\sqrt{5})/2 + 1$. Thus

$$\frac{t'}{t} = \frac{c_{p+1} + 1}{c_p + 2} \geqslant \frac{3 + \sqrt{5}}{2} - \frac{7 + 3\sqrt{5}}{2(c_p + 2)}.$$

Hence, if $c_{p+1} \geqslant (c-1)(3+\sqrt{5})/2$ infinitely often, then $\sup(t'/t) \geqslant (3+\sqrt{5})/2$.

Otherwise, there exists an $s \geqslant 1$ such that $c_{p+1} < (c-1)(3+\sqrt{5})/2$ for all $p \geqslant s$. Choose $p \geqslant s$ such that $c_{p+1} - c_p \geqslant c_{q+1} - c_q$ for $0 \leqslant q < p$. Such a p must exists; if not, for $t - l > \max\{c_{q+1} - c_q \mid 0 \leqslant q < s\}$, the algorithm would never achieve an (l, t) stopping configuration.

Let $l = 0$ and let $t = c_{p+1} - c_p + 1$. Then, from Lemma 4.4, it follows that $t' = c_q + t - l$, where $q = \min\{j \mid j \geqslant 0,\ l \leqslant c_j$, and $t - l \leqslant c_{j+1} - c_j\} \geqslant p + 1$. Hence

$$\frac{t'}{t} = 1 + \frac{c_q}{t} \geqslant 1 + \frac{c_{p+1}}{c_{p+1} - c_p + 1}$$

$$= 1 + \frac{1}{1 - \dfrac{c_p - 1}{c_{p+1}}} > 1 + \frac{1}{1 - \dfrac{c_p - 1}{(c_p - 1)(3 + \sqrt{5})/2}} = \frac{3 + \sqrt{5}}{2}.$$

Therefore $\sup(t'/t) \geqslant (3 + \sqrt{5})/2$. ∎

As was the case with Theorem 4.3, Theorem 4.5 can also be generalized. In [3] it is shown that $\sup(t'/t) \geqslant (3 + \sqrt{5})/2$ for all cycle detection algorithms with two locations which have a fixed finite bound on the number of back advances they perform. Maria Klawe [10] recently showed that this lower bound in fact holds for all cycle detection algorithms which use two memory locations.

An algorithm which achieves this bound, the Fibonacci algorithm, is given in Fig. 4.3. Front moves are performed by this algorithm until $a - b$, the distance between the front and back markers, reaches Δ. At this point, the back marker jumps forward to join the front marker, Δ is increased in value, and the process repeats itself.

```
y ← x
b ← 0
z ← f(x)
a ← 1
Δ ← 1
while y ≠ z do
        begin
        if a − b = Δ then
                begin
                y ← z
                b ← a
                Δ ← Δ + a
                end
        z ← f(z)
        a ← a + 1
        end
```

FIG. 4.3. The Fibonacci algorithm.

THEOREM 4.6. *For the Fibonacci algorithm,* $\sup(t'/t) = (3 + \sqrt{5})/2$.

Proof. The analysis of this algorithm involves the Fibonacci sequence $\{F_p\}$, which can be defined as follows:

$$F_0 = 0, \qquad F_1 = 1, \qquad F_{p+1} = F_p + F_{p-1} \qquad \text{for all} \quad p \geqslant 1.$$

Let Δ_p denote the value of Δ after the block of three statements inside the **if** statement has been executed p times. It is easy to see that $c_p = F_{2p}$ and $\Delta_p = F_{2p+1}$ for $p \geqslant 0$.

In this proof, two inequalities concerning Fibonacci numbers are used: $(F_{2p+1})/(F_{2p}+1) < (1+\sqrt{5})/2$ and $(F_{2p+2})/(F_{2p+1}) < (1+\sqrt{5})/2$ for $p \geqslant 0$. These results are straightforward consequences of an alternate characterization of the Fibonacci numbers, namely,

$$F_p = \frac{(1+\sqrt{5})^p - (1-\sqrt{5})^p}{2^p\sqrt{5}}.$$

Suppose $0 \leqslant l < t$ and $c_p + 1 \leqslant t \leqslant c_{p+1}$. Then, by Lemma 4.4, $t' = c_q + t - l$, where $q = \min\{j \mid j \geqslant p,\ l \leqslant c_j,$ and $t - l \leqslant c_{j+1} - c_j\}$.

If $t - l \leqslant c_{p+1} - c_p$ and $l \leqslant c_p$ then $t' = c_p + t - l \leqslant c_{p+1}$ so that

$$\frac{t'}{t} \leqslant \frac{c_{p+1}}{t} \leqslant \frac{c_{p+1}}{c_p + 1}$$

$$= \frac{F_{2p+2}}{F_{2p}+1} = \frac{F_{2p}+F_{2p+1}}{F_{2p}+1} < 1 + \frac{F_{2p+1}}{F_{2p}+1} < \frac{3+\sqrt{5}}{2}.$$

If either $t - l > c_{p+1} - c_p$ or $l > c_p$, then $q = p + 1$. This is because $l < t \leqslant c_{p+1}$ and $c_{p+2} - c_{p+1} = F_{2p+4} - F_{2p+2} = F_{2p+3} > F_{2p+2} = c_{p+1} \geqslant t \geqslant t - l$. Thus $t' = c_{p+1} + t - l \leqslant c_{p+1} + t$.

Now $t - l > c_{p+1} - c_p$ implies $t > c_{p+1} - c_p$. In this case

$$\frac{t'}{t} \leqslant 1 + \frac{c_{p+1}}{t} < 1 + \frac{c_{p+1}}{c_{p+1} - c_p}$$

$$= 1 + \frac{F_{2p+2}}{F_{2p+2} - F_{2p}} = 1 + \frac{F_{2p+2}}{F_{2p+1}} < \frac{3+\sqrt{5}}{2}.$$

Finally, if $l > c_p$, then $t' = c_{p+1} + t - l < c_{p+1} - c_p + t$ so that

$$\frac{t'}{t} < 1 + \frac{c_{p+1} - c_p}{t} \leqslant 1 + \frac{c_{p+1} - c_p}{c_p + 1}$$

$$= 1 + \frac{F_{2p+2} - F_{2p}}{F_{2p}+1} = 1 + \frac{F_{2p+1}}{F_{2p}+1} < \frac{3+\sqrt{5}}{2}.$$

Thus $\sup(t'/t) \leqslant (3+\sqrt{5})/2$. By Theorem 4.5, $\sup(t'/t) \geqslant (3+\sqrt{5})/2$. Hence $\sup(t'/t) = (3+\sqrt{5})/2$. ∎

A closely related algorithm is the golden mean + 1 algorithm, depicted in Fig. 4.4. It too achieves $\sup(t'/t) = (3+\sqrt{5})/2$. The proof of this result is similar to that proof of Theorem 4.6 and can be found in [3]. For practical purposes, the Fibonacci algorithm is easier to implement since the only arithmetic operations it involves are integer addition and subtraction.

```
y ← x
b ← 0
z ← f(x)
a ← 1
while y ≠ z do
        begin
        if a ≥ ⌊((3 + √5)/2) b⌋ then
                begin
                y ← z
                b ← a
                end
        z ← f(z)
        a ← a + 1
        end
```

FIG. 4.4. Golden mean + 1 algorithm.

In [2], Brent discusses an algorithm B_ρ, parameterized by ρ, which also uses only two memory locations and performs no back advances. When $\rho = (3 + \sqrt{5})/2$, the algorithm is essentially the golden mean + 1 algorithm and when $\rho = 2$, the algorithm is a variant of Sedgewick and Szymanski's algorithm (Fig. 3.1).

5. SUMMARY

The lower bounds obtained for the various classes of cycle detection algorithms can be summarized as follows. The complexity of these algorithms is measured by $\sup(t'/t)$. This quantity relates the number of function evaluations performed by an algorithm to the smallest number used by any cycle detection algorithm, including those which use an unbounded amount of space.

Class of Algorithms	*Complexity*
M memory locations	$1 + (1/(M-1))$
bounded number of jumps, M memory locations	3
2 memory locations	$(3 + \sqrt{5})/2$.

These last two results are optimal.

Four algorithms were also discussed in this paper. The classes of algorithms to which they belong and the number of function evaluations they perform are

Algorithm	*Class*	*Complexity*
Sedgewick and Szymanski's algorithm	no back advances, M memory locations	$1 + (2/(M-1))$
Floyd's algorithm	no jumps, 2 memory locations	3
Fibonacci algorithm, golden mean + 1 algorithm	no back advances, 2 memory locations	$(3+\sqrt{5})/2$.

ACKNOWLEDGMENTS

I would like to thank Richard Karp for bringing this problem to my attention and for many valuable discussions. Shafi Goldwasser, Maria Klawe, Mike Luby, Silvio Micali, Joan Plumstead, and Vijay Vazarani patiently helped me clarify the exposition. I am also grateful to Micah Beck and the referees who pointed out an error in the proof of Theorem 2 in the earlier version of this paper [3].

REFERENCES

1. E. ALLENDER, "An Improved Lower Bound for the Cycle Detection Problem," Research Report GIT-ICS-83/4, School of Information and Computer Science, Georgia Institute of Technology, 1983.
2. R. P. BRENT, An improved monte carlo factorization algorithm, *BIT* **20** (1980), 176–184.
3. F. E. FICH, Lower bounds for the cycle detection problem, *in* "Proceedings of the 13th Annual ACM Symposium on Theory of Computing," pp. 96–105, Milwaukee, Wisconsin, 1981.
4. W. KAHAN, personal communication.
5. D. E. KNUTH, "Seminumerical Algorithms, The Art of Computer Programming," Vol. 2, Addison–Wesley, Reading, Mass., 1969.
6. N. PIPPENGER, personal communication.
7. J. M. POLLARD, A monte carlo method for factorization, *BIT* **15** (1975), 331–334.
8. R. SEDGEWICK AND T. G. SZYMANSKI, The complexity of finding periods, *in* "Proceedings of the 11th Annual ACM Symposium on Theory of Computing," pp. 74–80, Atlanta, Georgia, 1979.
9. R. SEDGEWICK, T. G. SZYMANSKI, AND A. C. YAO, The complexity of finding cycles in periodic functions, *SIAM J. Comput.* **11** (1982), 376–390.
10. M. KLAWE, personal communication.